intel

ARTICLE
REPRINT

AR-97

MAY 1979

Minimizing Risk Through the Use of Development Systems

by Eric Michelman

reprinted from **1979 Electro Professional Program**

INTRODUCTION

The difficulty and expense of developing production quality software has historically been greatly underestimated. This is particularly true for microcomputer systems where the state of software development is still comparatively young. The vast majority of such software is coded in assembly language, often by engineers with little training in effective programming techniques or software project management. Furthermore, the closeness to the hardware environment introduces an additional set of potential difficulties that must be considered.

In a practical sense, there are many areas of risk and expense in microcomputer software development. Some of it is the actual coding, but the majority of it is not. Far greater risks are presented by activities such as testing and debugging, maintenance, training, figuring out what the hardware really does, etc.

Microcomputer development systems offer several advantages in minimizing the risk and expense of software development. As a lab tool designed to provide support for all phases of development, it is well suited to provide a firm base for systematic and orderly project development. This is the key to effectively using development systems for minimizing risk — taking advantage of a comprehensive set of tools designed to support the entire development cycle in a consistent manner.

As a brief overview, the tools provided by a development system include:

- For system performance and programmer productivity, a variety of different programming languages for application to separate functional modules within the total application program.
- Also for system performance and project productivity, the easy combination of software modules coded by different programmers.
- Test and debug tools tailored to microcomputers, including in-circuit emulation or ICE™ for hardware/software debugging.
- For project management, software backup and documentation facilities.
- For training and maintenance, a consistent user interface for all the tools, and a complete set of support documentation.
- For overall confidence in meeting the project schedule, a commitment of vendor support for the entire development process.

CONTROLLING THE RISKS

The primary risks and expenses have been stated above. Next we will examine each of them in more detail. Note that we lump risks and expenses together because they are very much related. After all, the bottom line of all the risks involved is added expense, either in outright development cost, product cost, or the cost of delayed project completion/product introduction.

Project Schedules and Delays

Perhaps the most prevalent risk in software development (as in hardware development) is that the planned project schedule is not met. There are several common causes for project schedules not being met. Often it is simply because the schedule was not realistic — software development time is notorious for being estimated over-optimistically. Other factors, though, can make a real difference in meeting (or failing to meet) the schedule. Programmer productivity is the most obvious factor, with others such as good software project management also being important.

Development systems can have a tremendous impact on programmer productivity or, stated in another way, programming effectiveness. This is affected both by offering effective development tools and by combining them in a comprehensive "user-oriented" package. Simply stated, this means easy to use, powerful features, and above all, flexibility.

Development tools which are effective in increasing programming productivity include:

a) Advanced System Support — The basic tools of microcomputer development are similar to those for larger system development. A disk based operating system for efficiently saving and retrieving programs and data files is necessary. Console I/O (CRT and keyboard) provides an efficient human interface with the system. A comprehensive operating system program interface provides easy access to the system's functions and carries out many of the operational details programmers shouldn't and don't want to be bothered with.

b) High-Level Programming Languages — High-level languages such as PL/M, FORTRAN, and PASCAL offer far higher programmer productivity than assembly language. Differences in the range of 6-10 times are often quoted. The penalty paid is some loss of efficiency in terms of code size, depending on the language.

c) Library Facilities — Software library facilities are used to share common routines between programs, programmers, and applications. This requires relocation and linkage capabilities.

Duplication of programming efforts is minimized and, in a production environment, this is useful in project management.

d) Microcomputer-Oriented Debug Tools — By minimizing the time spent debugging code, programmer productivity will obviously increase. Debug tools tailored specifically to microcomputer software are discussed in the section below on debugging.

Good project management is largely a skill that comes with experience. While development systems do function as useful aids in project management, they remain just that — aids — they don't offer any ready-made solutions. Useful aids provided though include library facilities, convenient backup and documentation facilities, and often very good training materials and facilities. (See section below on training and employee turnover.)

Software Size and Performance

Closely related to meeting schedules is meeting product specs. With software this generally means code size and performance. With microcomputer software this is often particularly important since available memory is generally restricted and timing is often dictated by external devices.

It must be recognized, of course, that software performance is limited by the hardware environment, limited memory being the most common example. On the other hand, one wants to be sure to take full advantage of the hardware capabilities available.

A development system won't help pick the right chip, but it can provide the most effective software development aids available to support the selected chip.

By offering a range of software tools, the most effective tool can be used for each part of development. Where code size and speed are most important, an assembler should be used. Where structure, modularity, and easy maintenance are important, a block structured language such as PL/M or PASCAL should be used. Where built-in capabilities such as mathematical functions and formatted I/O are needed, an applications-oriented language like FORTRAN can be used. Relocation and linkage facilities are then used to combine the different object modules into one executable module.

Debugging

One of the more frustrating risks of software development is finishing the initial coding on time and then having the debugging drag well past scheduled completion. Aside from the emotional wear and tear, this can be very costly economically.

Development systems prove to be particularly effective in providing powerful and systematic debugging facilities.

By far the most powerful debugging tool in resident microcomputer development systems is in-circuit emulation, or ICE, a capability pioneered by Intel to provide real-time hardware/software debugging. Since ICE takes the place of the application CPU in communicating directly with the hardware, both the software and the hardware can be debugged as one unit. Further, conditional breakpoints can be set in the code and software and hardware conditions may be examined when breaks are reached. With ICE, the system may be run in real-time, or the user may choose single-step or multiple-step executions. Breakpoints may be set in the software and on hardware conditions.

ICE is used in debugging in two primary ways, first as a conventional symbolic software debug utility, and second as a hardware/software debug utility. This second capability is particularly important for microcomputer systems as it eliminates the situation where you debug the software, debug the hardware, and then when you put them together find out that you still have a lot of debugging to do. Further, this can proceed before the hardware is available for checkout — ICE resources such as memory, the CPU, and I/O can be used in place of the application hardware.

High-level languages also aid in debugging. Programs are easier to read and follow. With modular, structured programming techniques, errors are confined to individual modules and thus can be easily located.

Employee Turnover/Training/Maintenance

From a project viewpoint, a major risk is the departure of the engineers. This can have several effects, all of them bad:

1. The project will be delayed while new personnel are trained.
2. The project will be greatly delayed if partially completed software turns out to be unusable due to poor documentation.
3. When this happens after development is complete, the same problems apply to maintenance.

Development systems aid in training new employees by encouraging good coding techniques (and thus more readable code), presenting

a consistent, easy-to-use user interface, and a high level of vendor support.

The question of vendor support is a vital one. The primary training materials generally come from the vendor and in-depth training courses are often available.

When using a development system from the component manufacturer, such as Intellec® Microcomputer Development Systems from Intel™, one gets an extra degree of support as the vendor has the added commitment of actively supporting the development process. Further, the synergy produced by receiving development and component support from the same people is substantial. Not only is there a comprehensive body of knowledge to draw on, but the responsibility for supporting the entire process is clearly accepted.

On the negative side, there is, as with any complex system, a significant amount of learning that may be necessary for full use of the more advanced development systems. While this is to be expected in using these powerful tools, it can be a drawback in situations with rapid employee turnover.

CONCLUSION

The effective use of microcomputer development systems can provide substantial benefits in microcomputer software development. More specifically, they can be very useful in minimizing many of the risks inherent in software development projects. Of particular interest is the full level of support throughout the development effort. This includes a wide variety of software tools, comprehensive software and hardware debugging facilities, and full service support from the vendor.

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 • (408) 987-8080

Printed in U.S.A./H-197/1179/10K/TP BL